1992-05

Multiple-valued programmable logic array minimization by simulated annealing

Dueck, Gerhard W.

Multiple-valued programmable logic array minimization by simulated annealing," Proceedings of the 22nd International Symposium on Multiple-Valued Logic, May 1992, pp. 66-74

Multiple-Valued Programmable Logic Array Minimization by Simulated Annealing*

Gerhard W. Dueck†, Robert C. Earle†, Parthasarathy Tirumalai‡, and Jon T. Butler†

†Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5004

‡Hewlett-Packard Laboratories
Bldg. 3U-7, 1501 Page Mill Road
Palo Alto, CA 94304

Abstract

We propose a solution to the minimization problem of multiple-valued programmable logic arrays (PLA) that uses simulated annealing. The algorithm accepts a sum-of-products expression, divides and recombines the product terms, gradually progressing toward a minimal solution. The input expression can be user-specified or one produced by another heuristic.

Unlike recently studied minimization techniques (which are classified as direct-cover methods), our technique manipulates product terms directly, breaking them up and joining them in different ways while reducing the total number of product terms. We show two mechanisms for recombining product terms and compare the results with presently known heuristics. A benefit of simulated annealing is that improved solutions can be achieved by increasing computation time.

1: Introduction

The only known algorithm for finding a minimal sum-of-products expression is exhaustive search. However, excessive computation time makes this approach impractical. For example, in a comparison of minimization algorithms on simple expressions [11], three days of computation time were required to produce minimal expressions by exhaustive search, while only three seconds were required for heuristic algorithms to produce *near* minimal expressions.

Sum-of-products expressions are interesting because of the ease with which they can be implemented by programmable logic arrays (PLA's). Implementation by PLA is easier than by random logic because the circuit designer needs to provide only a design for the row-column intersection. Recent progress in the implementation of multiple-valued PLA's has occurred in CCD [5]. Implementations have been proposed for current-mode CMOS [15].

Because of the computational complexity associated with minimal sum-of-products solutions, there is considerable interest in heuristics. At least four are known; Pomper and Armstrong [7], Besslich [1], Dueck and Miller [4], and Yang and Wang [14]. All use the direct cover method; that is, first a minterm is selected and then an implicant is chosen that covers the minterm. This process is repeated until the given expression is covered. Using search in conjunction with the direct cover method [15], improves the realizations but increases the computation time. The increased computation time has inspired research into parallel minimization algorithms [12, 13].

We propose an alternative to the direct cover method. Instead of creating implicants, our algorithm manipulates existing implicants. That is, implicants are combined, reshaped, or divided. Manipulation of implicants is not new; it is used in binary minimization [2], and was proposed for multiple-valued sum-of-products expressions [8]. What is new is the means of manipulation. We do it nondeterministically. That is, randomly chosen implicants are randomly combined, reshaped, or divided. The number of implicants in a cover of an expression increases when an implicant is divided and during certain reshapings, which allows one to go from a local minimum to a global minimum. The algorithm is essentially a series of transitions from solution to solution in the solution space. As time goes on, the probability decreases that a divide or reshape that increases the number of product terms is performed. In so doing, the transitions among solutions become gradually biased towards solutions with fewer product terms.

*Research supported in part by the Naval Research Laboratory, Washington, DC through direct funds at the Naval Postgraduate School, Monterey, CA and by a National Research Council Research Associateship tenured at the Naval Postgraduate School, Monterey, CA.

The process suggested above is similar to the slow cooling of metals or glass, which allows the "melt" to reach a low energy state (a crystalline state). This is termed *annealing*, and the corresponding optimization method is called *simulated annealing*. *Slow* cooling is essential to the achievement of a minimal solution. On the contrary, in both the physical system and the optimization model, rapid cooling or *quenching* yields nonminimal results. With sufficiently slow cooling, simulated annealing can provide practical solutions to many optimization problems [6]. It has the further advantage that if certain conditions hold [9], the probability of achieving a global minimum approaches 1.0. This is unlike deterministic heuristic algorithms, in which nonminimal solutions can occur; indeed, our experience [11] is that it is easy to find expressions for which a given heuristic does *not* achieve the minimal solution. The achievement of a global minimum with a probability that approaches 1.0 may not be satisfactory if the probability is low after a reasonably long computation. However, our experience suggests that improved results over all known heuristics are obtained with reasonable computation times for large expressions.

2: The sum-of-products minimization problem

An r-valued function, $f(x_1,x_2,\cdots,x_n)$, takes on a value from $\{0,1,...,r-1\}$, for each assignment of values to the variables, which are also r-valued; i.e. $x_i \in \{0,1,...,r-1\}$. r, the radix, is the number of logic values in the system. Because of their widespread use, we choose to represent a function in its sum-of-products form. Because the truncated sum is so easily implemented in multiple-valued PLA's, we choose to use it. A *product term* or *implicant* is expressed as

$$c\ {}^{a_1}x_1^{b_1}\ {}^{a_2}x_2^{b_2}\ \cdots\ {}^{a_n}x_n^{b_n}, \qquad (1)$$

where $c \in \{1,2,...,r-1\}$, is a nonzero constant, where the literal function,

$${}^{a_i}x_i^{b_i} = r-1 \text{ if } a_i \le x_i \le b_i\ ,$$

$$t = 0 \qquad \text{otherwise.}$$

and where concatenation is the *min* function; i.e. $x\,y = \min(x, y)$. Since the literal function ${}^{a_i}x_i^{b_i}$ takes on only values 0 and $r-1$, the product (*min*) of literals ((1) without c) is either 0 or $r-1$, while the complete term (with c) takes on values 0 and c. Indeed, (1) is c iff $a_i \le x_i \le b_i$, for *all* i. For two variable functions, it is convenient to represent a product term as a rectangle of values of c extending from $x_1 = a_1$ to $x_1 = b_1$ and from $x_2 = a_2$ to $x_2 = b_2$.

The sum-of-products expression for any function $f(x_1,x_2,\cdots x_n)$ is some number of product terms summed together using the truncated sum operation, shown as a + on the left-hand side of

$$a + b = \min(r-1, a+b)\ ,$$

where the + on the right-hand side is ordinary addition with logic variables viewed as integers. Thus, if this sum should exceed $r-1$, the *min* operation will assign $r-1$ to the logic expression $a + b$.

Any function $f(x_1,x_2,\cdots,x_n)$ can be represented in a sum-of-products form; for example, each assignment of values to the variables that yields a nonzero value for $f(x_1,x_2,\cdots,x_n)$ can be represented as a product term that is 0 for all other assignments of values. Such a product term is called a *minterm*. Summing all minterms, using the truncated sum operation, yields the function $f(x_1,x_2,\cdots,x_n)$.

A minimal sum-of-products expression is any expression with the minimal number of product terms. While the problem of finding one sum-of-products expression for a function is straightforward, as shown in the example immediately above, the problem of finding a *minimal* expression is another matter altogether. The *sum-of-products expression minimization problem* is to find a sum-of-products expression with the fewest product terms for a given function $f(x_1,x_2,\cdots,x_n)$. As stated in the introduction, the only known algorithm for solving this is exhaustive search.

3: Simulated annealing

Simulated annealing has been introduced [6] as a means to solve large-scale optimization problems. It is based on a principle in statistical mechanics, in which a low energy crystalline state is achieved by first melting a substance and then slowly cooling it.

3.1: The general process

A basic computation in simulated annealing is a *move*. A move is a transition from one solution to another solution in the solution space. In the sum-of-products minimization problem, the solution space is the set of all sum-of-product expressions for some given function. Associated with each solution is a cost, which, for the case at hand, is the number of product terms. Intuitively, one favors moves that decrease the cost, since this drives the system to a minimal solution. However, in a solution space with local minima, the exclusive application of cost-decreasing moves can produce nonoptimal solutions. Cost-increasing or *hill-climbing* moves are also needed if the system is to

recover from a local minimum.

In simulated annealing, prospective moves are chosen at random. If a move decreases the cost, it is always accepted. If it increases the cost, it is treated probabilistically. That is, a cost-increasing move is accepted with probability $P(\Delta E) = e^{-\Delta E/T}$, where T is the temperature, and ΔE is the increase in cost as the result of making the move. Initially, a high temperature is chosen, in which case $P(\Delta E)$ is high. Here, almost all moves are accepted, regardless of whether they increase or decrease the cost. A system that is held in this state for a sufficiently long time is considered to be *melted*. In the melted state, there is no progress toward a minimal solution; rather the system undergoes random changes and is typically far from a minimal solution.

Once the system has persisted in this state, the temperature is reduced and the process repeated. The probability of accepting a cost-increasing move is now slightly lower. This process continues, as the decreasing temperature gradually decreases the probability of accepting cost-increasing moves. The result is slow progress toward an optimal state. Eventually, the system reaches a point where there is no further improvement. The system is considered to be *frozen*.

The temperature reduction process is called the *annealing schedule*. It is critical to the attainment of a global minimum. When the temperature is rapidly reduced, a process called *quenching*, the result is often far from optimal [6]. Therefore, a slow decline is preferred, even though this requires more computation time. A typical annealing schedule, and one that we use, is described by

$$T_n = \alpha T_{n-1},$$

where α is between 0.80 and 0.99. Here, the temperature at each stage is a large (but constant) fraction of its former value. Values of α less than 0.80 are considered quenching in our designs.

A move in the minimum sum-of-products minimization problem is a two-step process. First, a pair of product terms is randomly chosen. Second, a test is applied to determine if they are equivalent to a single product term. If so, they are replaced by the equivalent product term. Otherwise, they are replaced by a set of two or more product terms. We describe these two steps in the following subsections.

3.2: Choosing a pair of product terms

Two completely separate product terms cannot be combined, and such product terms are not considered. The algorithm considers only adjacent product terms, selecting one at random. Two product terms are *adjacent* if and only if a minterm of one is either coincident or adjacent to a minterm of the other. For example, Fig. 1 below shows a function with four pairs of adjacent product terms, 1-2, 3-4, 5-6, and 7-8. Two product terms *combine* if they can be replaced by a single product term. For example, of the four adjacent product terms in Fig. 1, three combine. The pair 3-4 combine to the single product term 3 (3 is said to *absorb* 4), 5-6 combine because these are equivalent to a single product term consisting of a pair of (horizontal) 2's, and 7-8 combine because these are equivalent to a single product term consisting of a pair of (vertical) 2's.

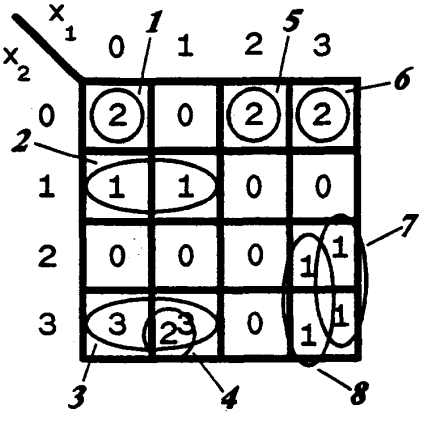


Figure 1. Example of a Function to Be Minimized.

3.3: Operations on a pair of product terms

We consider two moves, cut-or-combine and reshape. Both choose a pair of product terms, as described above. While both combine the pair in the same way, each executes the replacement in different ways. Our motivation in investigating two types of moves is the insight gained on how the efficiency of simulated annealing depends on the sophistication of the move.

3.3.1: The cut-or-combine move

In the *cut-or-combine* move, the two product terms are combined, if possible, as explained above. If not, a cut is performed, as follows. One of the two products is chosen randomly with probability 0.50. If the chosen product term is a minterm of value 1, the move is rejected and another pair of adjacent product terms is chosen. However, if the chosen product term is not a 1 minterm, it is divided. A division can occur along the logic value, in which case, two product terms are formed each with the same literals as the original product terms, but with coefficients that sum to the

coefficient of the original product term. For example, in a 4-valued system, if a product term with coefficient 2 is cut, it is replaced by two product terms each with coefficient 1. If the original product term coefficient is $r-1$, then the coefficients of the divided terms can be anything as long as their truncated sum is $r-1$. A product term divided in such a way is said to undergo a *logical* divide. For example, in a 4-valued system, if a minterm with coefficient 3 is cut, it can be replaced by two minterms each with coefficient 2.

A product term can also be divided *geometrically*. In this case, a variable value with a literal range of two or more is chosen. Then, two product terms with the coefficient of the original product term but divided along that variable are chosen. In Fig. 1, product terms 5 and 6 resulted from a geometric divide, while 7 and 8 resulted from a logical divide. Of all ways to divide a product term logically and geometrically, one is chosen with uniform probability.

3.3.2: The reshape move

The cut-or-combine move is basic. It provides a fundamental cost-increasing move, the cut, where a single product term is converted into two product terms. It also provides a cost-decreasing move, the combine, where two product terms are converted into a single product term. The *reshape* move, like the cut-or-combine, operates on a randomly chosen pair of product terms. Also like the cut-or-combine, it combines the two product terms if a combine is possible. However, for two product terms that cannot be combined, the reshape move proceeds differently.

First, the *consensus* operation is applied. If the two product terms overlap, the consensus of the two product terms is a product term situated at the intersection of the two terms with a coefficient that is the truncated sum of the coefficients of the two product terms. If the two product terms do not overlap, then they must be disjoint, (but strictly adjacent). In this case, the consensus is a single product term that is a part of both terms with a coefficient that is the minimum of the coefficient of the two product terms. The part of each product term that contributes to the consensus of the two is the "face" of the intersection that extends along the whole of the variable across which the two product terms are adjacent.

Fig. 2 shows an example of the two subcases of the consensus operation. The consensus is indicated by the hatched area. Fig. 2a shows the consensus in the case of overlapping product terms, while Fig. 2b shows the consensus in the case of nonoverlapping product terms. For each of the two product terms, the consensus is subtracted. Of what is left, there are several ways to divide the remaining product terms. Fig. 2 shows one way. From the ways that result in the fewest product terms, one is chosen randomly. Unlike the cut-or-combine move, the reshape move can produce more than three product terms from the original two. Indeed even two different product terms can result; such moves are done at no cost. An example of the latter occurs in the case of product term 2 in Fig. 1 and product term 1 replaced by a 1 minterm. The application of the reshape move yields a product term consisting of a vertical pair of 1's plus a 1 minterm. In this example, the consensus term is the vertical pair of 1's, $1\ {}^{0}x_1^{0}\ {}^{0}x_2^{1}$.

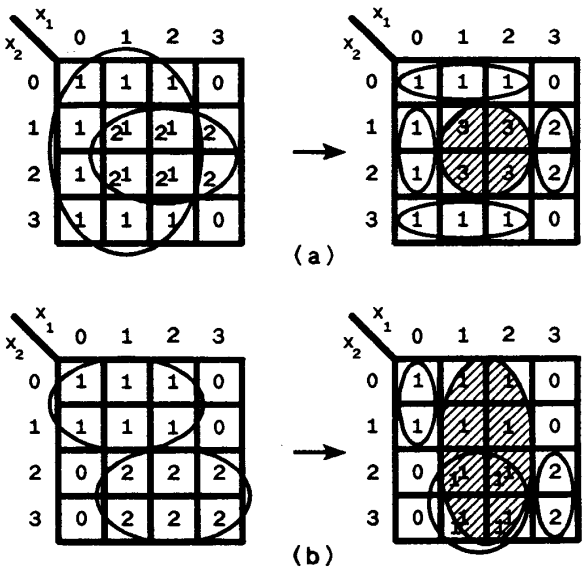
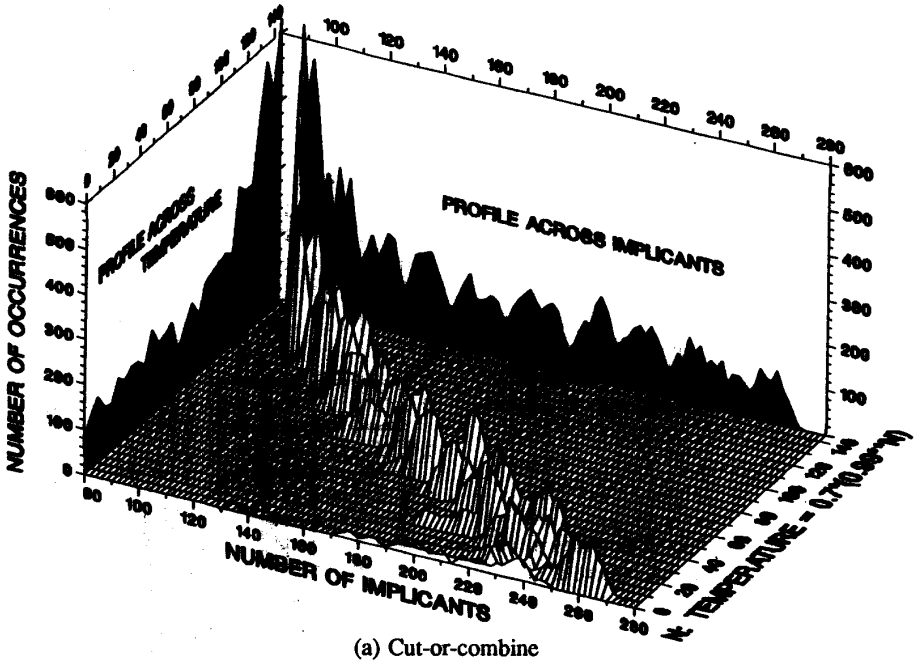


Figure 2. Example of the Consensus Operation.

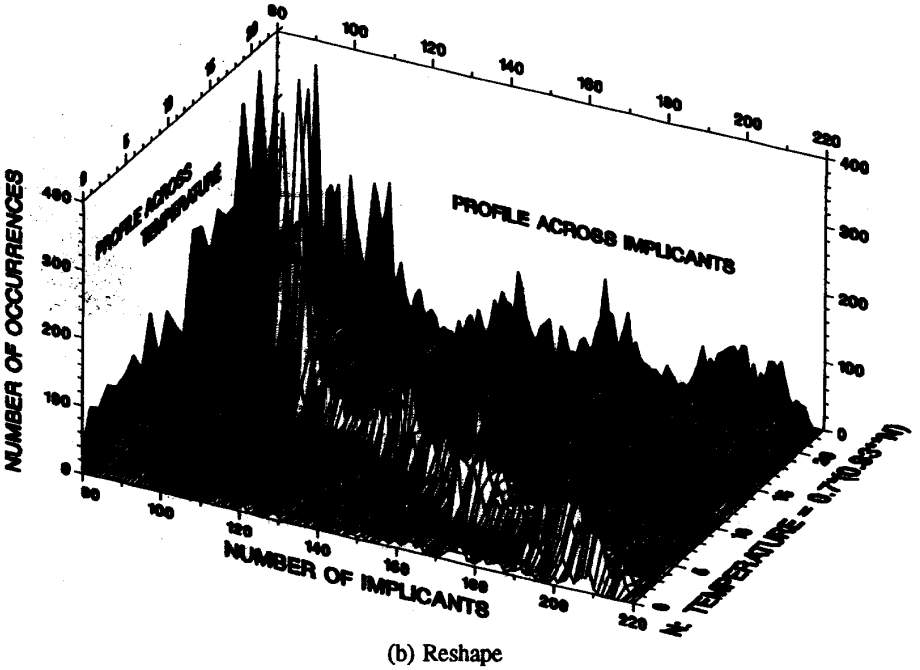
The reshape move suffers from a disadvantage. For example, in a 4-valued system, consider a minimal sum-of-products expression consisting of two product terms with coefficient 2 in the form of a cross. A 3 occurs at the intersection. Given an initial solution consisting of five disjoint parts of the cross, there is no sequence of moves that will allow the reshape move to achieve the minimal solution. That is, the reshape move, while able to form one half of the cross, is unable to form the other half. The best solution found by reshape is with three implicants. Unlike the cut-or-combine move, the reshape move does not create product terms that oversum. In this example, as in others, this ability is necessary to achieve a minimal solution.

4: Experimental results of simulated annealing

Unlike previous analyses, which considered expressions with few product terms, we consider, in this section, an expression with significantly more product terms. Fig. 3 shows the result of simulated annealing



(a) Cut-or-combine



(b) Reshape

Figure 3. Simulated Annealing Using the (a) Cut-or-combine and the (b) Reshape Moves.

using the two types of moves, cut-or-combine (a) and reshape (b) applied to a randomly chosen 4-variable 4-valued function with 200 minterms. Prior to the annealing process, this function was minimized using the Dueck and Miller heuristic [4], resulting in a solution of 96 product terms. In both graphs, the number of product terms is plotted horizontally with larger numbers to the right. The temperature is plotted in the axis perpendicular to the page, with higher temperatures in the front. The number of times a visit is made to a solution with some number of product terms is plotted along the vertical axis. Vertical "slices" represent a histogram of the number of times the system is in a solution with the corresponding number of product terms specified along the horizontal axis. Each slice represents one temperature. The slice in the very front represents the highest and starting temperature. It shows how melting takes place. For this temperature, moves quickly transform the initial 96 product term solution into solutions with approximately 275 product terms. At the temperature just higher than the melted state, almost all of the solutions have nearly the same number of product terms, and the vertical deviation is larger.

It is interesting that the total number of product terms in the melted state is greater than the number of minterms. In the cut-or-combine move, there are solutions in the melted state that have approximately 275 product terms. This exceeds the 200 nonzero minterms in the initial specification of the expression because of a property of the cut-or-combine move: Given a product term, it is possible to cut it into two product terms identical to the initial product term except that the coefficients of the latter sum to the coefficient of the former. Indeed, if the coefficient of the initial product term is $r-1$, there can be many ways the sum can occur because of oversumming (e.g. when $r = 4$, there are five ways to form the sum of 3, versus only one way to form the sum of 2).

A similar phenomenon occurs with the reshape move. For example, consider a two-variable 4-valued function consisting two 3's at the opposite corners of a 2 by 3 rectangle with a pair of adjacent 2's in between. This function can yield five product terms by a sequence of reshape moves starting from four minterms. That is, starting with three product terms, two 3 minterms and the pair of 2's, there is a sequence of reshape moves that will produce a five product term solution where each 3 minterm is replaced by a 2 and a 1 minterm.

Once the melted state has been reached, there is a gradual trend toward fewer product terms as the temperature decreases. The temperature axis is logarithmic. That is, every equally spaced slice represents a temperature that is some fraction α of the temperature of the slice just in front. In the case of the cut-or-combine move, α is 0.99, and in the case of the reshape, α is 0.93. As simulated annealing progresses, the slice in the front is generated, followed by the slice just behind it, etc.. The slow migration towards solutions with fewer product terms is evident. As the temperature decreases (moving toward the origin), there is a gradual shift to solutions with fewer product terms, until eventually all transitions are among solutions with very few product terms. In the case of cut-or-combine, a solution of 87 product terms is achieved, while in the case of the reshape, a solution of 84 product terms is achieved. It is interesting that cut-or-combine with a slower rate of temperature decline produced a solution with more product terms than the reshape. The values for α were chosen carefully to provide good solutions with reasonable execution times.

With the cut-or-combine, a total of 91.4 minutes of computation time were required on a Solbourne Series 4 workstation (equivalent to a Sun 110), while 3.98 minutes were required for the reshape. This illustrates the relation between the annealing schedule and the computation time. With $\alpha = 0.99$, the cut-or-combine exhibits a slower rate of decline in temperature than the reshape move, where $\alpha = 0.93$. Compensating for the large number of temperatures in the cut-or-combine move is the additional time required by the reshape to manipulate the product terms.

5: Comparison of simulated annealing with existing minimization algorithms

To achieve a fair comparison of simulated annealing with other heuristic minimization methods, we consider two types of test functions. The first consists of individual functions selected for their unique characteristics. The second consists of functions randomly generated by the HAMLET CAD tool.

In the first set, there are three functions. *Test1* is a randomly chosen 4-valued 3-variable function with 50 minterms. An exhaustive search in HAMLET shows that the exact minimal solution contained 21 product terms. The Dueck and Miller heuristic [4] in HAMLET results in a solution of 24 product terms. This is the form put into the simulated annealing program. After 104.4 minutes of computation time on a Solbourne Series 4 Workstation, cut-or-combine produced a (minimal) solution of 21 product terms, while reshape produced a solution of 21 product terms, but within 2.5 minutes. *Test2* is a 4-valued 4-variable symmetric function with 176 minterms and a minimal solution of 6 product terms. The minimal sum-of-products expression

for this is

$$1\,{}^{2}x_1^3\,{}^{2}x_2^3\,{}^{0}x_3^3\,{}^{0}x_4^3 + 1\,{}^{2}x_1^3\,{}^{0}x_2^3\,{}^{2}x_3^3\,{}^{0}x_4^3 +$$

$$1\,{}^{2}x_1^3\,{}^{0}x_2^3\,{}^{0}x_3^3\,{}^{2}x_4^3 + 1\,{}^{0}x_1^3\,{}^{2}x_2^3\,{}^{2}x_3^3\,{}^{0}x_4^3 +$$

$$1\,{}^{0}x_1^3\,{}^{2}x_2^3\,{}^{0}x_3^3\,{}^{2}x_4^3 + 1\,{}^{0}x_1^3\,{}^{0}x_2^3\,{}^{2}x_3^3\,{}^{2}x_4^3.$$

Its special characteristic is that it is difficult to minimize by cut-or-combine. That is, the minimal solution exists among many nonminimal solutions that are easily produced by the random cutting of product terms. The random nature of cut-or-combine makes it difficult to converge to the minimal solution from among the many nonminimal solutions. Reshape, on the other hand, tends to maintain group integrity, not introducing miscellaneous logical cuts that tend to move away from the minimal solution. *Test3* is a 4-valued 2-variable function that was chosen because reshape does not find the minimal solution for it. *Test3* is shown in Fig. 4.

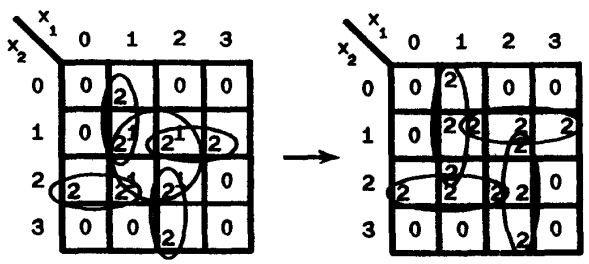


Figure 4. *Test*3, **a Test Function.**

This function requires oversumming where the truncated sum indeed truncates. Because of this, reshape does not achieve a minimal solution. It is relatively simple, and so cut-or-combine finds the solution easily. Table I shows the results of various algorithms on the three test functions. Our expectation of the relative merits of cut-or-combine and reshape on *Test2* and *Test3* are borne out. Interestingly, only the Dueck and Miller heuristic found the minimal solution on *Test2*, while cut-or-combine produced a solution quite far from optimal. For *Test3*, only the cut-or-combine and the heuristic by Yang and Wang [11] achieved the minimal solution.

The second group of tests consists of randomly chosen functions. For this test case, nine ensembles of ten functions each were chosen. Each ensemble consists of 4-valued 4-variable functions with the same number of minterms, a value that ranged from 50 to 250. Fig. 5 shows the results. Cut-or-combine performed best in ensembles having fewer minterms, while reshape had the best performance on the remaining functions. Fig. 6 shows the execution time of the various heuristics. This shows that the increased "intelligence" exhibited by reshape results in an improved solution, as well as reduced computation time. Both simulated annealing heuristics, on the average, outperformed the other heuristics. This improved performance is not without a price. Computation times are higher.

6: Concluding remarks

This investigation of the use of simulated annealing in finding minimal sum-of-products expressions has been encouraging. First, the time of computation is easily controlled; one can choose a slow annealing schedule, and, in so doing, achieve a solution that tends to be closer to optimum, or a fast schedule with less likelihood of achieving the optimum. Second, simulated annealing has general applicability, and there is the prospect of applying it to further problems in multiple-valued logic circuit design, e.g. layout and routing. Indeed, it may be represent the means to go on to stru-

Function	*Test1*			*Test2*			*Test3*		
	# implicants		seconds	# implicants		seconds	# implicants		seconds
Heuristic	**in**	**out**	**time**	**in**	**out**	**time**	**in**	**out**	**time**
Cut-&-combine	24	21	6264	14	19	2125.4	5	4	207*
Reshape	24	21	147	14	7	71.0	5	5	4.6
Dueck & Miller	24	24	0.9	14	6	5.7	5	5	0.03
Pomper & Armstrong	24	24	0.4	14	10	3.0	5	5	0.01
Yang & Wang	24	22	8.6	14	10	9.3	5	4	0.12

*This is the total time. The minimal solution was first found in 4.2 secs..

Table I. Number of Product Terms and Computation Time in Seconds for Three Test Functions as Produced by Five Heuristics.

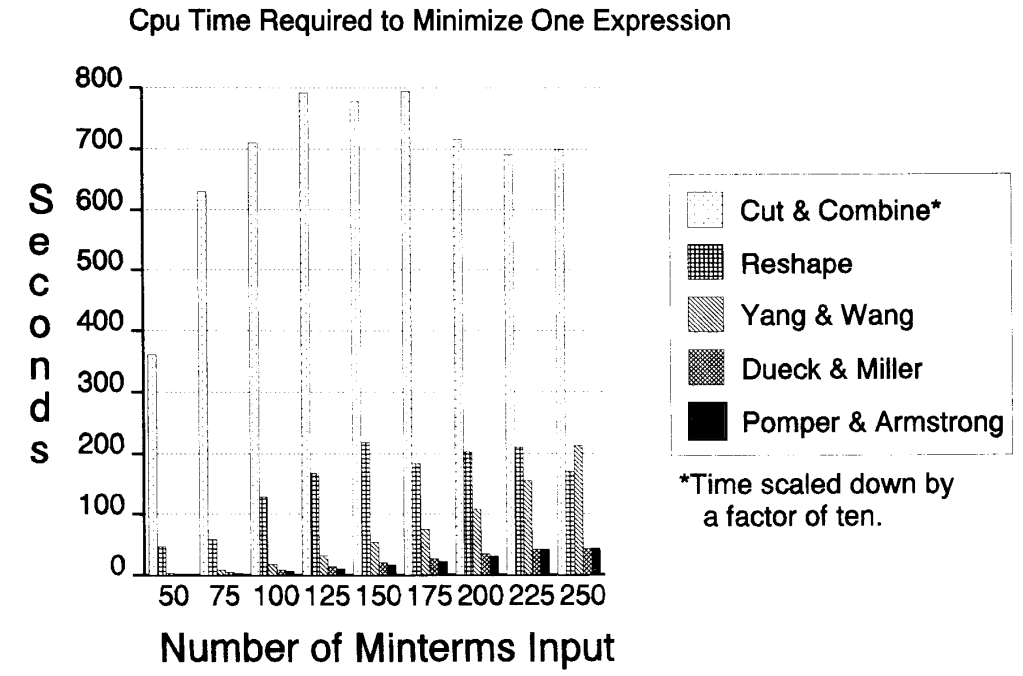


Figure 5. Comparison of Number of Product Terms Produced by Various Heuristics Versus the Number of Minterms Over Random Functions.

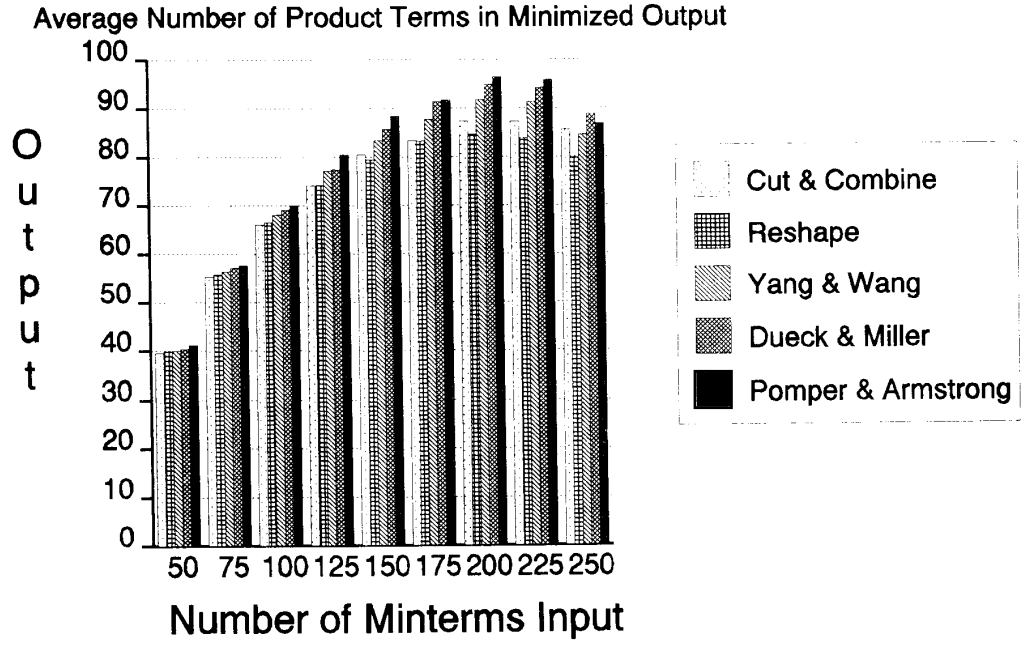


Figure 6. Comparison of Execution Time Required by the Various Heuristics Versus the Number of Minterms Over Random Functions.

tures more complex than PLA's, thereby achieving even more compact circuits.

We have shown two algorithms, the simple cut-or-combine and the reshape. The latter requires more computation time for an individual move, but yields good solutions with less computation time overall than the cut-or-combine. However, for expressions representing few minterms, the cut-or-combine move is superior. Both represent improvements to all known heuristics.

7: Acknowledgments

The authors thank the anonymous referees for their comments. An extended version of this paper appears in [3].

References

[1] P. W. Besslich, "Heuristic minimization of MVL functions: A direct cover approach," *IEEE Trans. on Comput.*, February 1986, pp. 134-144.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic, Boston 1984.

[3] G. W. Dueck, R. C. Earle, P. Tirumalai, and J. T. Butler, "Multiple-valued programmable logic array minimization by simulated annealing," Naval Postgraduate School Technical Report NPS-EC-92-004, Feb. 1992.

[4] G. W. Dueck and D. M. Miller, "A direct cover MVL minimization using the truncated sum," *Proc. of the 17th Inter. Symp. on Multiple-Valued Logic*, May 1987, pp. 221-227.

[5] H. G. Kerkhoff and J. T. Butler, "Design of a high-radix programmable logic array using profiled peristaltic charge-coupled devices," *Proc. of the 16th Inter. Symp. on Multiple-Valued Logic*, May 1986, pp. 100-103.

[6] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, No. 4598, 13 May 1983, pp. 671-680.

[7] G. Pomper and J. A. Armstrong, "Representation of multivalued functions using the direct cover method," *IEEE Trans. on Comput.* Sept. 1981, pp. 674-679.

[8] D. L. Ostapko, R. G. Cain, and S. J. Hong, "A practical approach to two-level minimization of multiple-valued logic," *Proc. of the Inter. Symp. on Multiple-Valued Logic*, May 1974, pp. 168-182.

[9] F. Romeo and A. Sangiovanni-Vincentelli, "Probabilistic hill climbing algorithms: properties and applications," ERL, College of Engineering, University of California-Berkeley, CA, March 1984.

[10] T. Sasao, "On the optimal design of multiple-valued PLA's," *Proc. of the Inter. Symp. on Multiple-Valued Logic* May 1986 pp. 214-223.

[11] P. P. Tirumalai and J. T. Butler, "Minimization algorithms for multiple-valued programmable logic arrays", *IEEE Trans. on Comput.*, Feb. 1991, pp. 167-177.

[12] P. P. Tirumalai and V. G. Vadakkencherry, "Parallel algorithms for minimizing multiple-valued programmable arrays," *Proc. of the 21st Inter. Symp. on Multiple-Valued Logic*, May 1991, pp. 287-295.

[13] C. Yang and O. Oral, "Experiences of parallel processing with direct cover algorithms for multiple-valued logic minimization," *Proc. of the 22nd Inter. Symp. on Multiple-Valued Logic*, May 1992.

[14] C. Yang and Y.-M. Wang, "A neighborhood decoupling algorithm for truncated sum minimization," *Proc. of the 20th Inter. Symp. on Multiple-Valued Logic*, May 1990, pp. 153-160.

[15] J. Yurchak and J. T. Butler, "HAMLET - An expression compiler/optimizer for the implementation of heuristics to minimize multiple-valued programmable logic arrays", *Proc. of the 20th Inter. Symp. on Multiple-Valued Logic*, May 1990, pp. 144-152.